# Detection of Obfuscated Mobile Malware with Machine Learning and Deep Learning Models

K. A. Dhanya[1](✉), O. K. Dheesha[2], T. Gireesh Kumar[1], and P. Vinod[2]

[1] TIFAC CORE in Cyber Security, Amrita School of Engineering, Amrita University, Coimbatore, India
dhannyashibu@gmail.com, t_gireeshkumar@cb.amrita.edu
[2] SCMS School of Engineering and Technology, Cochin, India
dheeshaprakash@gmail.com, vinodp@scmsgroup.org

**Abstract.** Obfuscation techniques are used by malware authors to conceal malicious code and surpass the antivirus scanning. Machine Learning techniques especially deep learning techniques are strong enough to identify obfuscated malware samples. Performance of deep learning model on obfuscated malware detection is compared with conventional machine learning models like Random Forest (RF), Classification and Regression Trees (CART) and K Nearest Neighbour (KNN). Both Static (hardware and permission) and dynamic features (system calls) are considered for evaluating the performance. The models are evaluated using metrics which are precision, recall, F1-score and accuracy. Obfuscation transformation attribution is also addressed in this work using association rule mining. Random forest produced best outcome with F1-Score of 0.99 with benign samples, 0.95 with malware and 0.94 with obfuscated malware with system calls as features. Deep learning network with feed forward architecture is capable of identifying benign, malware, obfuscated malware samples with F1-Score of 0.99, 0.96 and 0.97 respectively.

**Keywords:** Obfuscated malware detection · Machine learning · Deep learning · Random forest · Classification and regression trees · K nearest neighbor

## 1 Introduction

The significance of mobile phone is countless as it offers a variety of incredible features and opportunities. The tremendous progress in the field of mobile technology together with availability and access to the internet has resulted in an innovative experience in mobile computing. This has been made possible by developing mobile applications. Many factors contributed to the massive growth of mobile applications. Mobile applications are increasingly playing a vital role in the commercial world. A large number of business applications have sprung up with the smart phone industry boom. Some of the most popular types of business applications are banking apps, online shopping apps, mobile payment apps and communication apps. Cyber terrorism is growing exponentially in a sophisticated manner, such that it is mandatory to employ a strong anti-malware strategies to protect the information and digital assets. As per the IT threat evolution Q3 statistics 2018, Kaspersky Lab solutions clogged around 947,027,517 attacks initiated from online assets [1]. McAfee Global Threat Intelligence examined further 1,800,000 URLs, 800,000 files, and 200,000 files in a sandbox each day [2]. Security is becoming more extortionate and hard to manage. The organizations around the world have started spending whopping amounts of money to protect their software from these attacks. Worldwide cyber security spending is said to have reached 96 billion dollars in 2018 [3].

Malware authors have resorted to techniques like obfuscation to evade detection by anti-malware system. Obfuscation has numerous genuine uses, including making software secure and preventing tamper. It also plays an important role in aiding malware to evade different detection mechanisms. Obfuscation is a technique that models programs difficult to interpret. It converts the code to a new version. Originally, this technology was aimed at protecting the intellectual property of software developers. Software developers may also employ obfuscation techniques to conceal flaws and vulnerabilities of the code. Code obfuscation in malicious context has two goals: the malware must elude detection and outlive long enough to accomplish its tasks. Conventional antivirus use signature based or pattern based malware detection. Malware writers can easily defeat the antivirus by changing the syntax without changing the malicious semantics using obfuscation techniques [5]. So it is very crucial to identify obfuscated malware.

Deep learning models are significant in android malware detection since it improves the classification performance. Semantics hidden in the sequence of features can be easily identified by this models [27]. Obfuscated malicious behaviour can be easily revealed with deep learning layers. This work make the following contribution in the area of obfuscated mobile malware detection: Performance of Deep learning models are better than conventional machine learning models for identifying obfuscated malware. Performance of dynamic feature (system calls) is better than static feature (hardware and permission) for conventional machine learning models. Obfuscation transformation attribution requires sophisticated pattern mining methods other than association rule mining with apriori algorithm. Rest of the paper is organised as follows. Section 2 covers literature survey on existing obfuscated malware detection techniques. Section 3 describes architecture and theoretical background of machine learning and deep learning models. Experimental results are demonstrated and analysed in Sect. 4. Section 5 concludes the work.

## 2 Related Work

Signature-based detection in an anti-malware approach can be easily evaded using simple obfuscation techniques [4]. Static analyses alone are not capable enough to identify malware that incorporates evasion mechanisms. The anti-malware tools that correctly identifies malware failed to detect them after the transformation mechanisms were applied to it. The techniques adopted by malware writers are becoming increasingly sophisticated. This proposes the need for re-designing the malware detection methods so as to effectively protect the smart devices. Numerous experimentation and analysis has been carried out to assess the performance of antivirus tools against obfuscated android mobile malware application. It is observed that many anti-viruses failed to detect obfuscated applications. A detailed literature survey for identifying obfuscated mobile malware given in Table 1.

**Table 1.** Related work

| Author | Methodology and inferences |
|---|---|
| M. Ikram et al. 2019 [7] | Features from weighted directed graph of API calls. Machine learning models: SVM, KNN, Random Forest. 96% of malware samples were correctly classified |
| O. Mirzaei et al. 2018 [5] | Features: Dalvik bytecode, control flow graph. Classification models: KNN, SVM, Decision Trees, Random Forest. Accuracy of 92.2% for identifier renaming, 81.41% for string encryption, 68.32% for control flow obfuscation and overall accuracy of 80.66% for the combination of more than one obfuscation technique |
| G. Suarez-Tangil et al. 2017 [8] | Extra trees are used for malware detection and family identification. Features (API calls, permission, code structure, invoked components, native components, obfuscation artifacts, invariant features underobfuscation). Malware detection accuracy - 99.82% and family identification accuracy - 99.26% |
| Y. Wang et al. 2017 [9] | Support vector machine for provenance analysis. Features are extracted from data section of DEX files. Obfuscator dentification accuracy - 97%, Configuration recognition accuracy - 90% |
| J. Garcia et al. 2016 [10] | API based features are used. SVM is used for malware detection and CART is used for family identification. Malicious app detection accuracy - 98% and family identification accuracy - 95% |

Most of the works in obfuscated malware analysis concentrates on evaluating the strength of antimalware software on obfuscation techniques and evaluating existing malware detection methods on obfuscation transformation. Obfuscation detection in mobile applcation using batch learning not able to compete in terms of accuracy but save computational resources and time. Androdet [5] claims efficient obfuscated malware detection system. But the authors of Androdet failed to account for the fact that their dataset is biased [6]. As a result machine learning models fail to learn a generalist model for string encryption and might instead learn to classify samples based on characteristic of each malware family. When Androdet is reevaluated with samples never appeared in both the training and testing data, accuracy dropped to around 50%.

## 3 Methodology

A novel architecture is proposed for identifying obfuscated malicious android application. The model is evaluated with two static features and a dynamic feature. The static features considered here are permission and hardware and dynamic is system call. Figure 1 illustrates the architecture of the proposed model for obfuscated android malware detection.

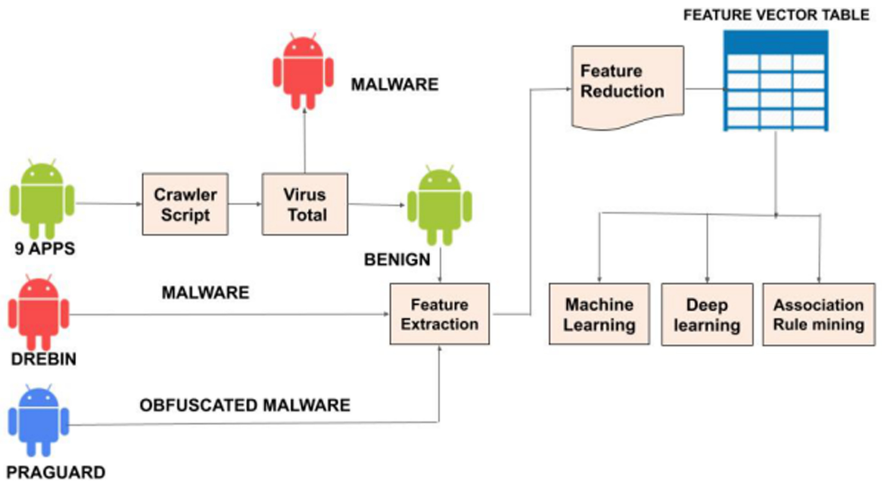

**Fig. 1.** Proposed system architecture

### 3.1 Problem Formulation

Let dataset (B, M, O) consist of Benign (B), Malware (M), and Obfuscated Malware (O) samples. Obfuscated Malware samples (O) are generated by applying different obfuscation strategies ($\Delta$) on Malware (M).

$$O = \Delta(M) \tag{1}$$

Each samples are represented as n + 1 tuples such that

$$X_{ij} = (x_{i1}, x_{i2}, x_{i3} .........x_{im}, C_i) \tag{2}$$

where $x_{ij}$ represent the value of $j^{th}$ feature of $i^{th}$ sample and $C_i$ is the class label of sample $X_i$, $C_i \in$ (B, M, O). Problem is to build a three class classification model capable of mapping the class label of Y$(y_1, y_2, ....y_n)$ to B, M, O.

## 3.2 Dataset

The dataset for the proposed system consists of three types of samples-benign, malware and obfuscated malware. A total of 5750 benign samples are collected from the Google Playstore [11] and their genuineness is verified using Virus Total [12]. 5000 malware samples are collected from Drebin [13] and 5000 obfuscated samples are collected from PRAgard dataset [14].

## 3.3 Feature Extraction

– Static features: Two static features are considered: Permission and hardware features. They are extracted using the Android Asset Packaging Tool [15].
– Dynamic Feature: The dynamic feature considered here is system call. It is extracted using the Android Dumb Bridge [16] and the stress test is performed using the Monkey-runner [17].

## 3.4 Feature Reduction

Let $S$ be the set of features. Recursive Feature Elimination [18] will generate all combination of features and find $X \in S$ which produce best accuracy with logistic regression. For finding $X$ this approach considers all the subsets of S. For each subset $S_j$ classification model $M_{S_j}$ is generated by Logistic Regression (LR) [19],

$$M_{S_j} = LR(S_j)$$

and its performance is measured. Feature set for which the model produces the maximum performance is selected as $X$.

$$X = Max(Performance(M_{S_j})|S_j \in (subset(S))) \tag{3}$$

## 3.5 Feature Vector Table Construction

Classification Model $M$ is generated from the feature vector table $F$. Feature vector table is a $m \times n$ matrix where $m$ represents the number of samples and $n$ represents the count of the attributes. Feature occurrence matrix is generated for permission and hardware features, whereas frequency matrix for system call features.

Three classification models are generated for identifying obfuscated malware. They are Random Forest [20], Classification and Regression Trees [21] and K-Nearest Neighbour [22].

## 3.6 Deep Neural Network

Deep learning networks are strong enough to execute feature engineering and thereby can identify relevant features that correlate and then combine features to promote efficient malware detection. In this work a feed forward architecture is proposed with 3 layers with Rectified Linear Unit (ReLU) as activation function in hidden layer and sigmoidal activation function in the output layer. Feed forward network is trained to predict output for all the three features. In order to fit the given data on the feed forward network, the weights of the neural network are updated at the end of every iteration. The hyper-parameters [23,24], used for training deep neural network are epoch, dropout, batch and iteration. They are tuned to get optimised deep network that can identify malware and obfuscated malware efficiently by avoid over fitting of the data and eliminating vanishing gradient problem of deep network.

## 3.7 Evaluation Metrics

Metrics for evaluating proposed architecture is given in Table 2 [25].
Recall: Sensitivity of classification model is evaluated by recall and it measures the capability of the model to identify the real positive cases.
Precision: Confidence of the model is evaluated by precision. It denotes the true positive accuracy.
F1-Score: F1-Score is the harmonic mean of recall and precision.

**Table 2.** Evaluation parameters

| | |
|---|---|
| Precision | $\frac{TP}{TP+FP}$ |
| Recall | $\frac{TP}{TP+FN}$ |
| F1 score | $\frac{2*Precision*Recall}{Precision+Recall}$ |

## 3.8 Association Rule Mining

Obfuscation transformation attribution can be performed using association rule mining [26], which is strong enough to find features that are correlated and occur together. In this attribution model, apriori algorithm is used to perform association rule mining for reflection obfuscation with permission, hardware and system call features. Metrics for evaluating quality of mined rules are support, confidence and lift.

*Support:* Support of a feature $P$ is measured as the proportion of mobile Apps ($\alpha$) instances in which the feature appears.

$$Support(P) = \frac{\#\{P \in \alpha\}}{\#\alpha} \tag{4}$$

*Confidence:* Confidence says how likely a feature $Q$ will exist together with feature $P$. It is expressed as $P \rightarrow$ Q.

$$Confidence(P \rightarrow Q) = \frac{Support(P, Q)}{Support(P)} \tag{5}$$

*Lift:* Lift says how likely a feature $P$ occurs when a feature $Q$ is present.

$$Lift(P) = \frac{Support(P,Q)}{Support(P) * Support(Q)} \tag{6}$$

## 4 Results and Discussions

The experiment was conducted on Ubuntu 14.04 platform with the support of Intel core I7 and 8 GB RAM.

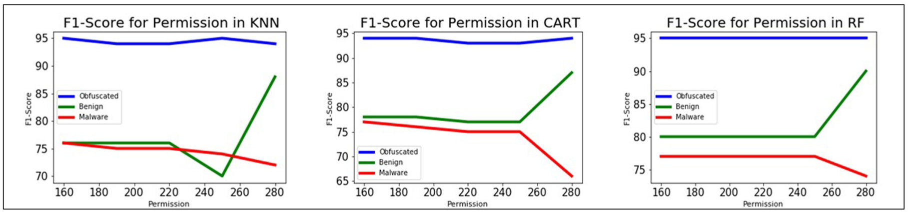

**Fig. 2.** Performance of classification algorithms with f1-score for permission

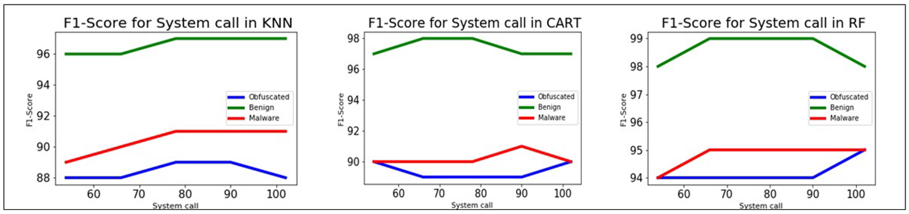

**Fig. 3.** Performance of classification algorithms with f1-score for system call

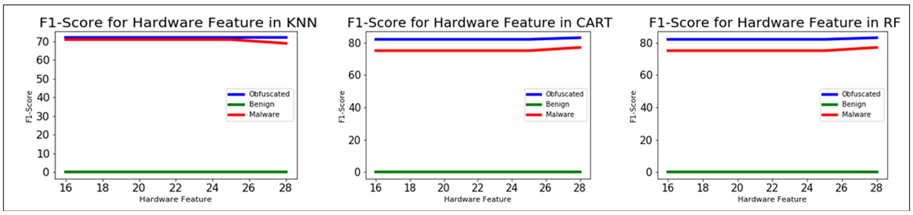

**Fig. 4.** Performance of classification algorithms with f1-score for hardware feature

## 4.1 Evaluation of Machine Learning Models

Classification models (CART, RF, kNN) with permission, hardware features and system calls are evaluated using three metrics which are recall, precision and F-Score and results are shown in Fig. 2, Fig. 3 and Fig. 4. Models are generated for different feature length: Permission (160, 190, 220, 250, 280), System Calls (54, 66, 78, 90, 102) and hardware features (16, 19, 22, 25, 28). Experimental results shows that Random forest outperforms other two classifiers with all the three features for malware and obfuscated malware detection due to its ensemble nature. Performance of the hardware features are not appreciable when compared with other two, as its fails to produce much variance among classes due to its limited size. Performance of the system calls are affected much by feature length. As feature length increases, F1-score improves and after a certain limit performance tends to decreases.

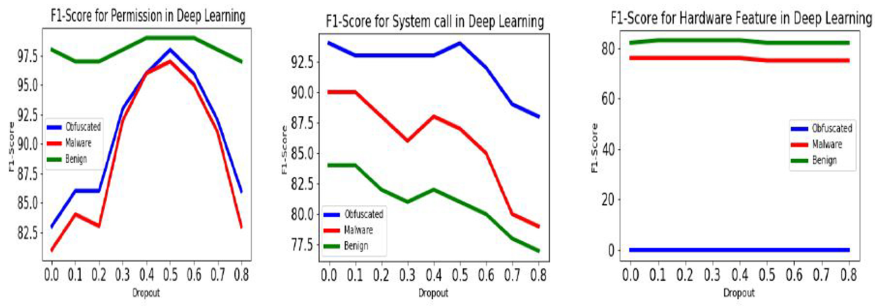

**Fig. 5.** Evaluation of deep neural network with f1-score

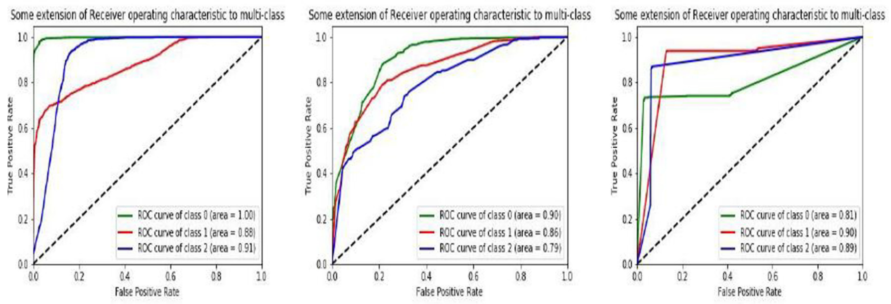

**Fig. 6.** ROC for permission, system call and hardware feature

### 4.2 Evaluation of Deep Neural Network (DNN) Models

The DNN is evaluated using recall and F1-score for permission, system calls and hardware features and results are shown in Fig. 5 and Fig. 6. F1-Score for malware and obfuscated malware in permission shows similar trends. F1-Score increases with increase in dropout rate till 0.5 with a value of 97.5 and then shows a gradual decrease. Benign samples show comparatively constant F1-Score values with no steep rises and falls. F1-Score decreases gradually for all the three classes in system calls with increasing dropout values. F1-Score for hardware feature for three classes are not correlated. Permission feature produces best Area under curve (AUC) for benign class and results shows permission and system call features performs much better than hardware feature. Permission features have produced 99% accuracy on deep learning model with dropout 0.5, which is comparatively greater than the existing DaDiDroid [7] model.

**Table 3.** Association rules

| Feature | Rule | Supp. | Conf. | Lift |
|---|---|---|---|---|
| Permission | android.permission.EXPAND_STATUS_BAR and android.permission.SET_WALLPAPER_HINTS | 0.02 | 1 | 34.7 |
| System calls | getuid32 and fcntl64 and clock_gettime | 0.10 | 1 | 4.72 |
| Hardware | gps and network | 0.35 | 0.88 | 2.5 |

### 4.3 Evaluation of Association Rule Mining

Performance of permission, hardware and system call features are evaluated in detecting reflection obfuscation using association rule mining is given in Table 3. Even though the support of permission based feature is poor, confidence factor is maximum for around 20 permissions which indicates the permissions are good for identifying reflection. Moreover highest lift value for around 20 permission is 34.785 which indicates these permissions are strongly bounded to reflection. Support value of system calls are better than permissions. Performance in terms of confidence score is same for all the three features considered. Lift value produced by system calls to identify reflection is less with maximum of 4.72 which depicts the poor associative nature of the system calls with reflection. Further performance of hardware features is less compared with permission and system calls with maximum lift value of 2.52.

## 5 Conclusion

Machine learning and deep learning models are evaluated for identifying obfuscated malware with features like permissions, hardware features and system calls. Random forest produced best results with permission and system call as it is a strong ensemble decision tree based classifier. Deep learning generates improved results with permission compared to conventional machine learning models. Scope of Association Rule Mining for obfuscation transformation attribution addressed in this work by generating rules for sophisticated reflection obfuscation. In future, various deep learning models can be applied and evaluated for obfuscation detection in malicious mobile application. Obfuscation transformations can be easily identified with resilient features that can be extracted using visualization techniques of malware source code which form the future scope to enrich this research work.

## References

1. Kaspersky Lab. https://securelist.com/it-threat-evolution-q3-2018-statistics/88689/. Accessed 4 May 2019
2. McAfee Labs Threats Report. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-sep-2018.pdf. Accessed 20 May 2020
3. Gartner Report. https://www.gartner.com/en/newsroom/press-releases. Accessed 15 Apr 2019
4. Scott, J.: Signature Based Malware Detection is Dead. Institute for Critical Infrastructure Technology, Illinois (2017)
5. Mirzaei, O., de Fuentes, J.M., Tapiador, J., Gonzalez-Manzano, L.: AndrODet: an adaptive Android obfuscation detector. Future Gener. Comput. Syst. **90**, 240–261 (2019)
6. Mohammadinodooshan, A., Ulf, K., Nahid, S.: Comment on "AndrODet: an adaptive Android obfuscation detector". arXiv preprint arXiv:1910.06192 (2019)
7. Ikram, M., Beaume, P., Kâafar, M.A.: DaDiDroid: an obfuscation resilient tool for detecting android malware via weighted directed call graph modelling. arXiv preprint arXiv:1905.09136 (2019)
8. Suarez-Tangil, G., Dash, S.K., Ahmadi, M., Kinder, J., Giacinto, G., Cavallaro, L.: DroidSieve: fast and accurate classification of obfuscated Android malware. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 309–320 (2017)
9. Wang, Y., Atanas, R.: Who changed you? Obfuscator identification for Android. In: 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), pp. 154–164. IEEE (2017)
10. Garcia, J., Hammad, M., Malek, S.: Lightweight, obfuscation-resilient detection and family identification of Android malware. ACM Trans. Softw. Eng. Methodol. (TOSEM) **26**(3), 1–29 (2018)
11. Google Play Store. https://play.google.com/store?hl=en. Accessed 25 Feb 2019
12. Virustotal. https://developers.virustotal.com. Accessed 25 Feb 2019
13. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C.E.R.T.: Drebin: effective and explainable detection of android malware in your pocket. In: NDSS, vol. 14, pp. 23–26 (2014)

14. PRAGard Dataset. http://pralab.diee.unica.it/en/AndroidPRAGuardDataset. Accessed 5 Mar 2019
15. Android Asset Packaging Tool. https://developer.android.com/studio/command-line/aapt2. Accessed 15 Mar 2019
16. Android Debug Bridge. https://developer.android.com/studio/command-line/adb. Accessed 15 Mar 2019
17. Android Monkey Runner. https://developer.android.com/studio/test/monkey. Accessed 15 Mar 2019
18. Gościk, J., Łukaszuk, T.: Application of the recursive feature elimination and the relaxed linear separability feature selection algorithms to gene expression data analysis. Adv. Comput. Sci. Res. **10**, 39–52 (2013)
19. Zakharov, R., Dupont, P.: Ensemble logistic regression for feature selection. In: Loog, M., Wessels, L., Reinders, M.J.T., de Ridder, D. (eds.) PRIB 2011. LNCS, vol. 7036, pp. 133–144. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24855-9_12
20. Biau, G.: Analysis of a random forests model. J. Mach. Learn. Res. **13**(1), 1063–1095 (2012)
21. Loh, W.-Y.: Classification and regression trees. Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. **1**(1), 14–23 (2011)
22. Cunningham, P., Delany, S.: K-nearest neighbour classifiers. Technical report. UCD School of Computer Science and Informatics (2007)
23. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
24. Nwankpa, C., Ijomah, W., Gachagan, A., Marshall, S.: Activation functions: comparison of trends in practice and research for deep learning. arXiv preprint arXiv:1811.03378 (2018)
25. Hossin, M., Sulaiman, M.N.: A review on evaluation metrics for data classification evaluations. Int. J. Data Min. Knowl. Manag. Process **5**(2), 1 (2015)
26. Agarwal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of the 20th VLDB Conference, pp. 487–499 (1994)
27. Alzaylaee, M.K., Yerima, S.Y., Sezer, S.: DL-Droid: deep learning based Android malware detection using real devices. Comput. Secur. **89**, 101663 (2020)